Estructura de Computadores:

Tema 4:
Entrada/Salida

Objetivos

- Conocer las caracteríscas particulares de los dispositivos que interactúan con el mundo exterior: dispositivos periféricos.
- Comprender los problemas de realizar intercambio de información con dichos dispositivos.
- Conocer los mecanismos que proporcionan los computadores para realizar sincronización: mecanismo de interrupciones.

Bibliografía recomendada

- Stallings, W. “Organización y arquitectura de computadores”. Prentice Hall. 7ª Edición. 2006.
- Patterson, D. A., Hennessy, J. L. “Computer Organization and Design”. Morgan-Kaufmann. 4ª edición. 2009.
- de Miguel, P. “Fundamentos de los computadores”. Paraninfo. 9ª edición. 2004

Índice

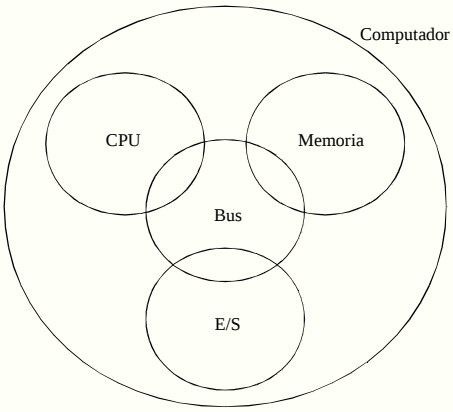
Entrada/Salida

Parte 1: Periféricos y Entrada/Salida programada

Índice

Periféricos

1. Introducción



Periféricos

- Ejemplos:
  - "Domésticos":
    - Ratón
    - Teclado
    - Impresora
    - Disco duro (HDD)
    - LANCE (Local Area Network Controller for Ethernet)
    - Etc., etc.
  - "Industriales"
    - Sensor de temperatura
    - Motores para la orientación de un telescopio
    - Sistema de control de actitud de un satélite artificial
    - Etc., etc.

Periféricos

- Gran diversidad:
  - Modo de funcionamiento
  - Formato y tamaño de los datos
  - Velocidad de transferencia
  - Tiempo de acceso
- Una posible clasificación:
  - Almacenamiento ➔ *Parallel I/O*
  - Comunicación:
    - Humanos ➔ Multimedia, *Brain interfaces*
    - Computadores ➔ Redes: *cluster, Grid computing*
    - "Medio físico" ➔ Sist. de control o *embedded systems*

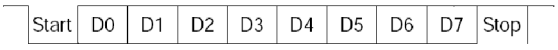
Periféricos

- Ejemplo 1: **Línea serie (UART)**
  - Dispositivo tipo carácter
  - Tamaño de los datos: buffer
  - V_{transf}
  - t_{acc}
  - Modo de funcionamiento:
    - Asíncrono/Síncrono (USART)
    - Paridad
    - Control de flujo

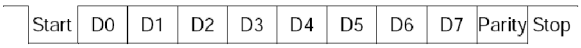
Ejemplo 1: UART

- A universal asynchronous receiver/transmitter (usually abbreviated UART and pronounced /ˈjuːɑrt/) is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with other communication standards such as EIA RS-232. (http://en.wikipedia.org/wiki/UART)

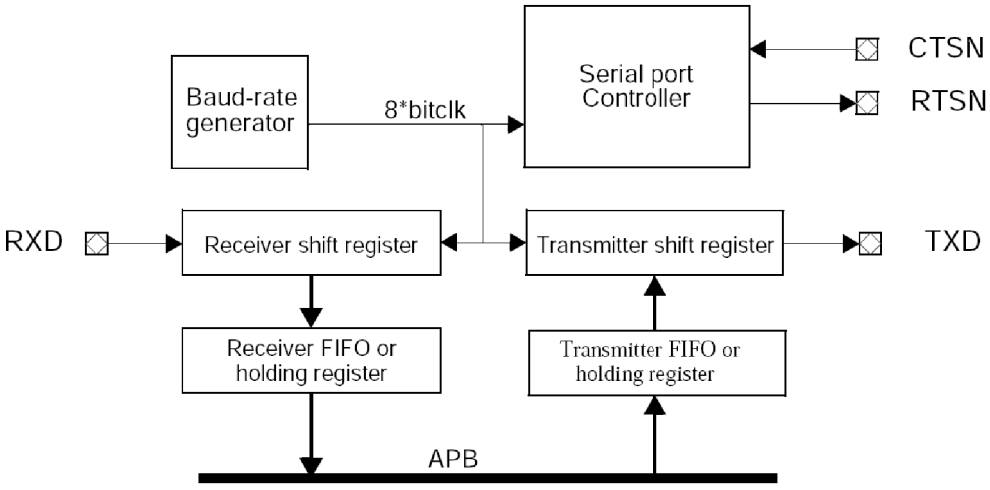
Data frame, no parity:



Data frame with parity:



Ejemplo 1: UART

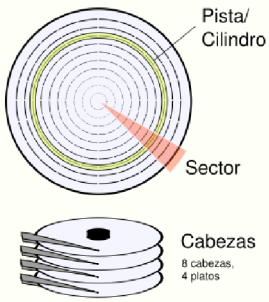


Periféricos

- Ejemplo 2: **Disco duro (HDD)**
  - Dispositivo tipo bloque
  - Tamaño de los datos: buffer
  - V_{transf}
  - t_{acc}
  - Modo de funcionamiento:
    - Organización: (p/c, sf, s)
    - Acceso a un sector
    - Distribución de múltiples sectores

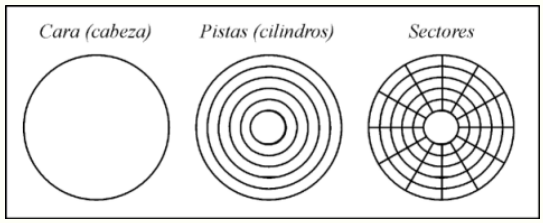
Ejemplo 2: HDD

- Nomenclatura: **disco duro** o **HDD** [*Hard Disk Drive*]
- Componentes:
  - Motor: p.ej, 7.200 r.p.m.
  - Discos: superficie magnetizable, p.ej., óxido de hierro
  - Cabezales: transductores electromagnéticos



Ejemplo 2: HDD

- Organización:
  - **superficies** (sf) o caras
  - **pistas** o **cilindros** (p/c)
  - **sectores** (s)

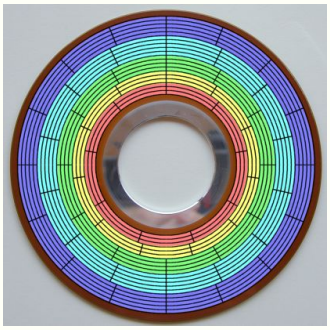


Ejemplo 2: HDD

- Parámetros:
  - **Capacidad**: p.ej., 500 GB
  - $\mathbf{V_{transf}}$: p.ej, 70 MB/s
  - $\mathbf{t_{acc}}$: p.ej, 5 ms
  - otras:
    - densidad de grabación lineal: bits/pulgada
    - densidad de grabación angular: bits/rad
- Otras:
  - distribución de los 'sectores lógicos' por cilindros
  - número variable de sectores/pista: *Zone Bit Recording (ZBR)*

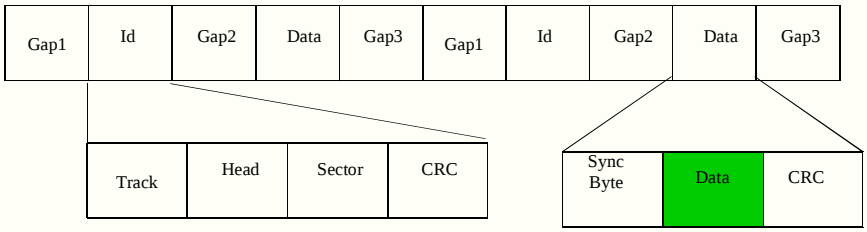
Ejemplo 2: HDD

- *Zone Bit Recording (ZBR)*
  - se utiliza habitualmente
  - su objetivo es aprovechar al máximo la superficie magnética
  - Vel. de transf.:
    - cte. dentro de cada zona
    - máxima en la zona más externa
    - mínima en la zona más interna
  - Densidad de grabación:
    - angular: fija/zona
    - lineal: fija/entre zonas



Ejemplo 2: HDD

- Capacidad bruta vs. capacidad neta
- Información bruta vs. información neta



Ejemplo 2: HDD

- Interfaces:
  - ST506:
  - ESDI
  - IDE
  - ATA
  - EIDE (FastATA),
  - SCSI
  - Etc.
- Medidas de fiabilidad:
  - MTBF: *Mean Time Between Failures*
  - Tecnología SMART: *Self-Monitoring Analysis and Reporting Technology*

Ejemplo 2: HDD

- Funcionamiento:
  - el motor gira siempre a la misma velocidad de rotación
  - el brazo se mueve hasta el cilindro destino: $\mathbf{t_{búsqueda}}$
  - una vez en cilindro destino, t giro hasta el comienzo del sector: $\mathbf{t_{latencia}}$
  - el t de transferencia será el de giro del sector: $\mathbf{t_{transf}} = t_{sect}$

$$t_{op} = t_{acc} + t_{transf}$$

$$t_{acc} = t_{búsqueda} + t_{latencia}$$

Ejemplo 2: HDD

Consideraciones sobre los tiempos que se deben tener siempre presentes:

- $t_{búsqueda}$:
  - el motor no se para, luego conforme la cabeza se mueve el disco habrá avanzado un determinado nº de sectores
- $t_{latencia:}$
  - en media es el tiempo de ½ vuelta
  - depende de la vel. de rotación y del nº del sector
- $t_{transf:}$
  - es el t de giro del sector e igual si es Lectura o Escritura
  - si no hay ZBR, $t_{transf} = t_{sect} = t_{rev}$/#sect/pista

Ejemplo 2: HDD

Ejemplo: HDD

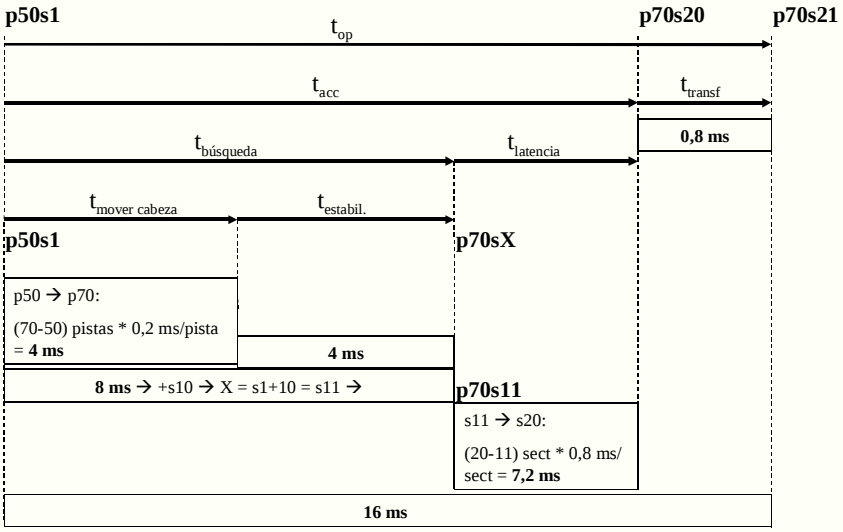
Características:

- Vel. de giro: 3.000 r.p.m. ➔ 20 ms/rev
- Nº de pistas: 500 pistas
- Nº de sectores/pista (fijos): 25 sect/pista ➔ t_{sect} = 0,8 ms/sect
- T de pista a pista consecutiva: 0,2 ms/pista
- T estabilización al llegar pista destino: 4 ms

Caso:

En t= 0s la cabeza del disco se encuentra al comienzo del sector s1 en la pista p50: ¿en qué instante concluirá la transferencia del sector s20 de la pista p70?

Ejemplo 2: HDD

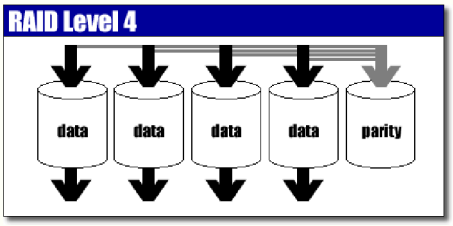


Ejemplo 2: HDD

RAID: *Redundat Array of Independent/Inexpensive Disks*

Objetivos:

- Incrementar la capacidad
- Mejorar el velocidad de transf. y/o el tiempo de acceso
- Aumentar la fiabilidad y la tolerancia a fallos
- Ejemplo: RAID 4



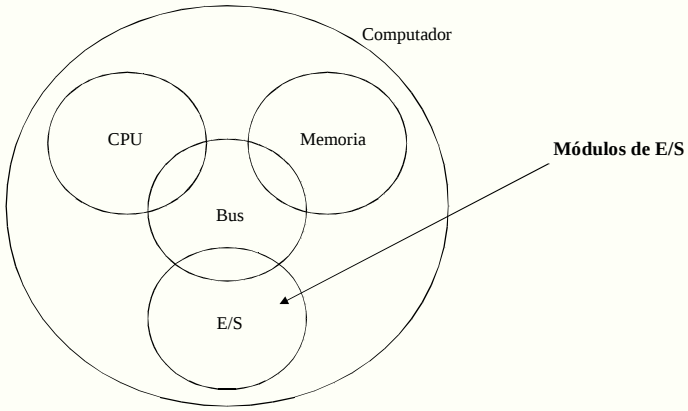
Problemática de la E/S

Gran diversidad de periféricos con características muy diferentes

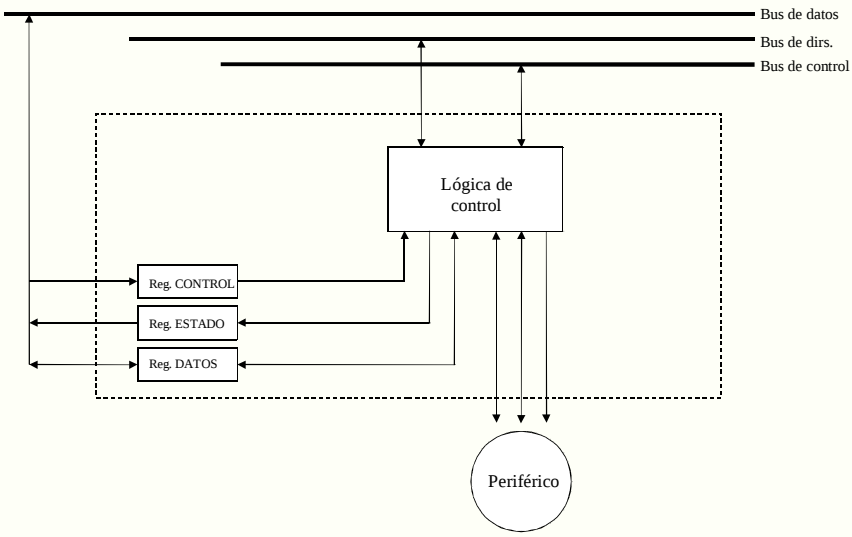
➔ es necesario “unificar” la visión Hw de los periféricos

➔***Módulos de E/S***

Módulos de E/S



Módulos de E/S



Instrucciones de E/S

- Instrucciones de la Arquitectura del computador para la transferencia entre los regs. de la CPU y los regs. de los Módulos de E/S:

```
OUT   .R1, /Dir_Reg1_Px; (orig → dest)
IN    .R1, /Dir_Reg2_Px;  (dest ← orig)
```

Direcciones de E/S

- Mapa de direcciones único: mismo ciclo de bus
  - Única línea de control MEMRQ (Memory Request) o AS (Address Strobe).
  - Las instrucciones de E/S son ld y st y se distinguen por la dirección.
- Mapas separados: nuevo ciclo de bus para acceder a direcciones I/O
  - Dos líneas de control: MEMRQ e IORQ (Input/Output Request).
  - Las instrucciones de E/S son in y out y se distinguen por su código de operación.

Decodificación de direcciones de E/S

- Direccionamiento geográfico: conjunto de direcciones prefijadas para cada ranura (slot)
  - Demasiado rígido para módulos de E/S
- Direccionamiento lógico: rango de direcciones configurable mediante interruptores en cada ranura
  - Laborioso y propenso a errores
- Bus PCI (Plug and Play):
  - Direccionamiento lógico configurable mediante registros de configuración.
  - Direccionamiento geográfico a los registros de configuración durante la iniciación.
  - Rutina inicial de configuración.

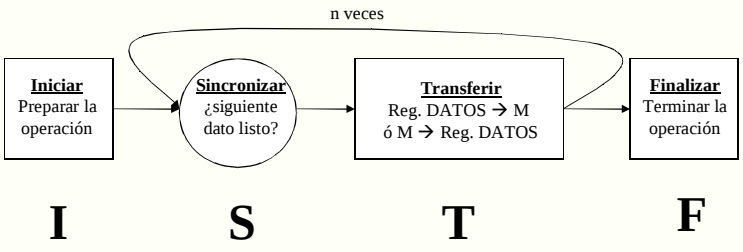
Técnicas de E/S

- **Técnicas de E/S**: grado de participación de la CPU en las operaciones de E/S

E/S programada
E/S por interrupciones
E/S por DMA

Técnicas de E/S

- **Operación de E/S**: transf. de un bloque de *n* palabras

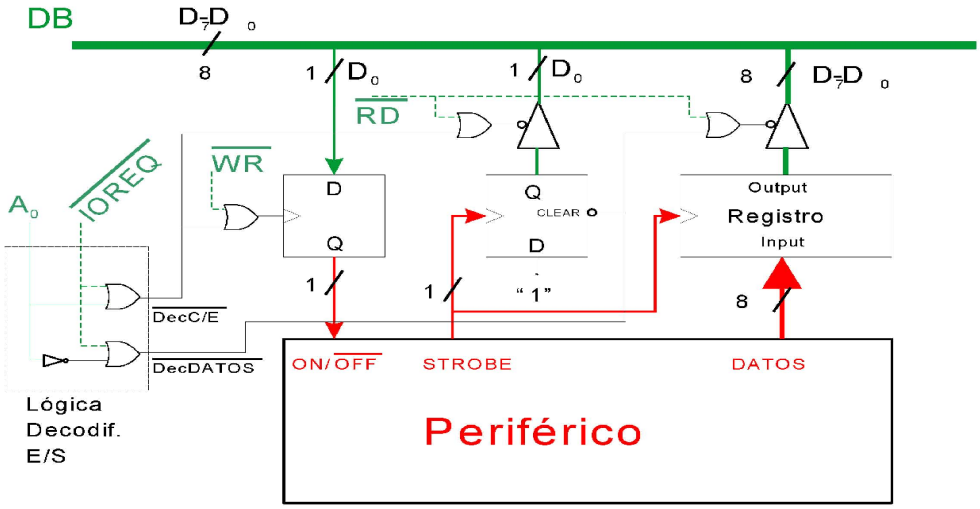


Técnicas de E/S

Técnica	Fase I	S	T	F
Programada	X	X	X	X
Interrupciones	X		X	X
DMA	X			X

E/S programada o directa

Módulo de E/S + *driver* para E/S programada



E/S programada o directa

Parámetros

- Tamaño del bloque: *n* palabras
- Dir. de almacenamiento en memoria: *dir_alm_M*

Direcciones de E/S

- Reg. de Control/Reg. de Estado: *Dir_C/E*
- Reg. de Datos: *Dir_Datos*

Mandatos (o 'comandos' [sic])

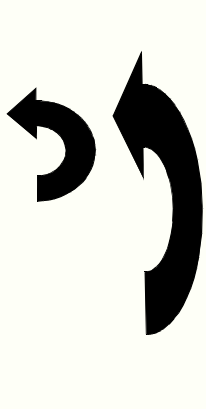
- Activar: *ON* [xxxxxxx1]
- Desactivar: *OFF* [xxxxxxx0]

Estado

- Nuevo dato listo: *LISTO* [00000001]

E/S programada o directa

```
        LD .R1,#dir_alm_M
        LD .R2,#n
        LD .R0,#H'01; ON
        OUT .R0, /Dir_C/E
sig:    IN .R0. /Dir_C/E
        CMP .R0, #LISTO
        BZ $sig
        IN .R0,/Dir_Datos
        ST .R0,[.R1++]
        DEC .R2
        BNZ $sig
        LD .R0,#H'00; OFF
        OUT .R0, /Dir_C/E
```



Entrada/Salida

Parte 2: Interrupciones y DMA

Índice

Índice

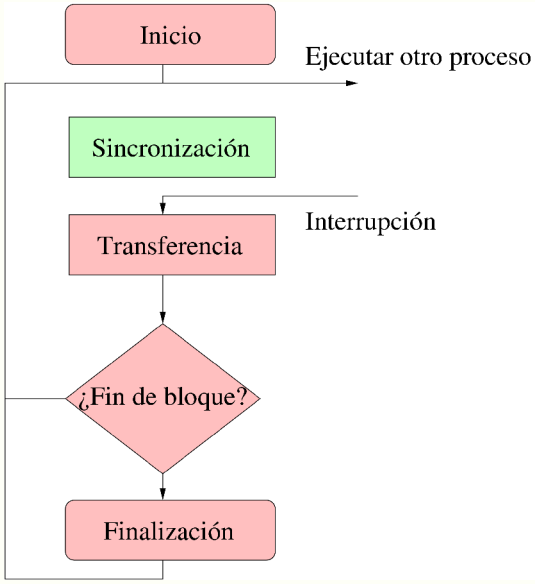
- Ejemplo de entrada/salida por interrupciones
- Operación con múltiples periféricos
  - Operación por muestreo *(polling)*
    - Identificación
    - Prioridades
    - Anidamiento
  - Operación mediante vectorización
    - Módulo de entrada/salida con vectorización
    - Secuencia de reconocimiento deinterrupciones
    - Esquema de prioridades hardware
      - Gestión centralizada
      - Gestión encadenada o *daisy chain*

Índice

    - Anidamiento de rutinas de servicio
      - Ejemplo
- Asignación de prioridades de interrupción
  - Interrupciones no enmascarables
- Conclusiones

- Entrada/salida mediante acceso directo a memoria *(DMA)*
  - Robo de ciclo aislado
  - Robo de ciclo en ráfagas
  - Módulo de entrada/salida con *DMA*

Entrada/salida por interrupciones



La CPU no se encarga de la sincronización

El módulo avisa a la CPU cuando está listo para una nueva transferencia

Se ahorra mucho tiempo de CPU

Interrupciones

- Suceso asíncrono que hace que la CPU no ejecute la instrucción apuntada por el PC
- Pasa a ejecutar la llamada rutina de servicio de interrupción
- Después ha de reanudar el programa interrumpido donde lo dejó
- Este mecanismo no puede afectar el comportamiento lógico de los programas

Interrupciones, excepciones y subrutinas

Suceso	Origen	Activación	Tratamiento
Subrutinas	Interno	Síncrona	Continuar
Excepciones	Interno	Asíncrona	Cancelar
Interrupciones	Externo	Asíncrona	Continuar

- El origen es el módulo de entrada/salida que tiene una temporización propia
- Pueden suceder en cualquier instante de la ejecución de una instrucción

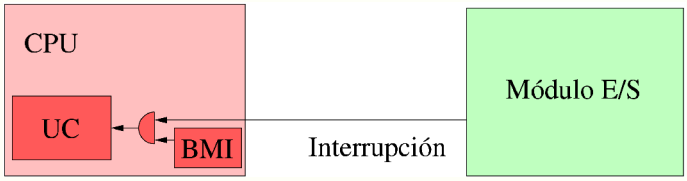
Solicitud de interrupciones

- Mediante una nueva señal a la unidad de control
- La unidad de control secuencia un conjunto de operaciones elementales para servir o tratar la petición de interrupciones

Servicio a peticiones de interrupción

- El servicio a interrupciones supone abandonar la ejecución del programa en curso y ejecutar otro programa que dé servicio a la solicitud del módulo de entrada/salida
- Posteriormente se ha de poder continuar con el programa interrumpido
- ¿Cuándo es posible hacer esto con el menor costo posible? Al finalizar la ejecución de la instrucción en curso ya que la CPU tendrá un estado consistente.

Biestable de máscara de interrupciones



- Hay dos programas ejecutando concurrentemente: el programa interrumpido y el que da servicio al módulo de entrada/salida -> condiciones de carrera
- No se puede impedir que el módulo pida interrupciones, pero sí que la unidad de control las “vea”

Instrucciones para habilitar e inhibir la atención a interrupciones

- Existen instrucciones para inhibir **DI *(Disable Interrupts)*** y habilitar **EI (*Enable Interrupts*)** la atención a las interrupciones
- BMI pertenece a la parte privilegiada del registro de estado (RE)
- Ejemplo familia x86:
  - CLI (Clear Interrpt Enable Flag) y STI (Set Interrpt Enable Flag)
  - IF (Interrupt Flag) es el bit 9 del Flags Register

Secuencia de reconocimiento de interrupciones

- La realiza la UC al final de cada instrucción si "ve" la línea de petición de interrupción activa
- Ha de salvar el estado necesario para posteriormente reanudar el programa interrumpido: no es un "cambio de contexto"
- La siguiente instrucción a ejecutar ha de ser la primera de la rutina de servicio de interrupciones

  Se hace antes de la secuencia de *fetch*

Secuencia de reconocimiento de interrupciones

```
FETCH: Si INT ∧ RE.BMI entonces:
          Salvar PC
          Salvar RE
          RE.BMI ← 1 (Inhibir interrupciones)
          RE.S ← 1 (Cambiar a modo privilegiado)
          PC ← DRTI (Dir. de la rutina de tratamiento de int.)
          ir a fetch
       Si no
          ir a fetch
```

- Además se inhiben las interrupciones para evitar reconocer eternamente la misma y se pasa a modo privilegiado para poder ejecutar instrucciones de entrada/salida

Instrucción de retorno de rutina de servicio de interrupción

```
RETI: Restaurar RE
      Restaurar PC
      ir a FETCH
```

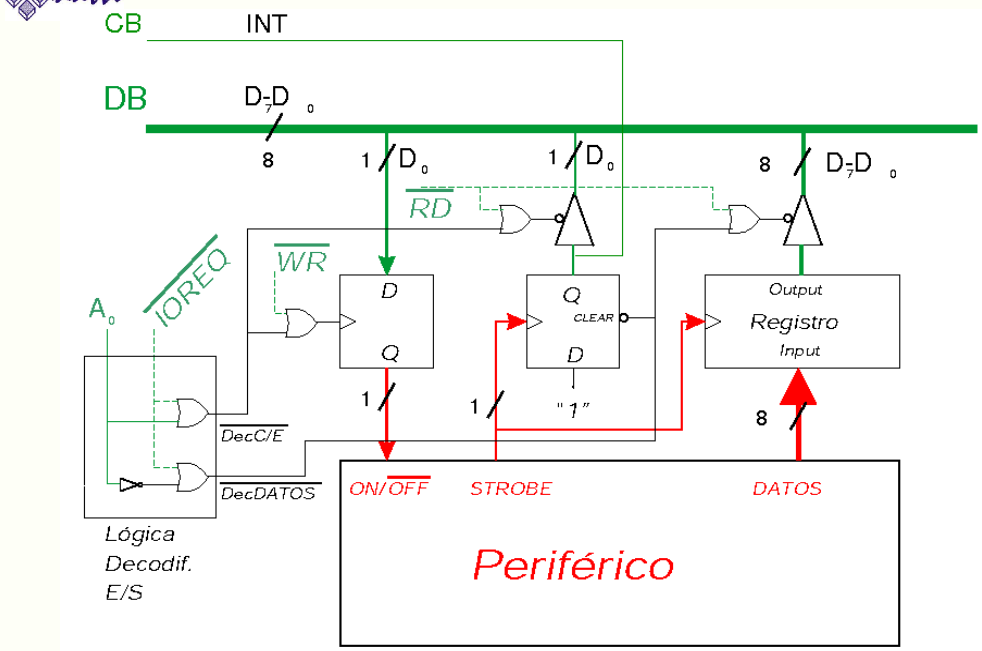
- Esta instrucción deshace lo hecho en la secuencia de reconocimiento de interrupciones
  - IRET (familia x86)
  - RETT (SPARC)

Rutina de servicio de interrupciones

- No debe alterar la lógica del programa interrumpido: debe **salvar y restaurar** el estado que altere
- Como se ejecuta debido a un suceso asíncrono y externo no se le pueden pasar parámetros ni por pila ni en registros sino en direcciones conocidas de memoria
- Finaliza con una instrucción RETI

Entrada/salida mediante interrupciones

Ejemplo con un periférico simple



Iniciación

```
LD .R1,#dir_alm_M
ST .R1,/Dir_dir_alm_M
LD .R2,#n
ST .R2,/Dir_contador
LD .R0,#H'01; ON
OUT .R0, /Dir_C/E
BR /Planificador
```

- `Dir_dir_alm_M` y `Dir_contador` contienen la dirección de memoria donde se almacenará el siguiente dato y el número de datos que quedan para completar la operación.

Rutina de servicio de interrupciones

```
PUSH .R0
PUSH .R1
PUSH .R2
LD .R1,/Dir_dir_alm_M
LD .R2,/Dir_contador
IN .R0,/Dir_Datos
ST .R0,[.R1++]
DEC .R2
CALLZ $Fin
ST .R1,/Dir_dir_alm_M
ST .R2,/Dir_contador
POP .R2
POP .R1
POP .R0
RETI

Fin:  LD .R0,#H'00; OFF
      OUT .R0, /Dir_C/E
      RET
```

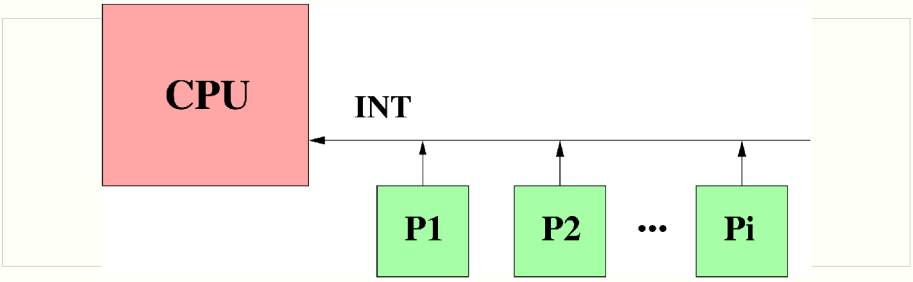

Entrada/salida mediante interrupciones

Operación con múltiples periféricos

Problemática

- Conexionado
- Identificación del solicitante
- Localización de la rutina de servicio correspondiente
- Prioridades en caso de peticiones simultáneas
- Anidamiento de rutinas de servicio

Conexión de varias periféricos



- La línea forma un OR cableado:
  - Se activa en el momento que alguno la activa y permanece activa hasta que todos la desactiven

Identificación mediante muestreo

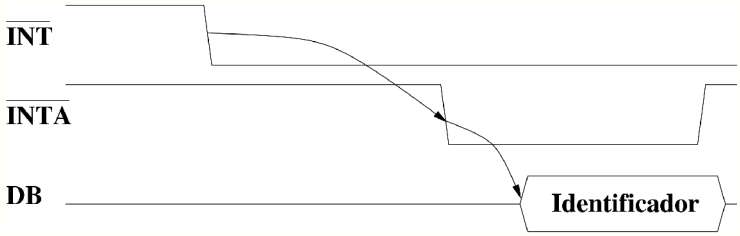
```
DRTI: PUSH    .R1
      IN      .R1, /Dir_C/E_P1
      CMP     .R1, #INT_Pendiente_P1
      BZ      $Rut_Espec_P1
      IN      .R1, /Dir_C/E_P2
      CMP     .R1, #INT_Pendiente_P2
      BZ      $Rut_Espec_P2
      ...

Rut_Espec_P1 Salvar estado
             Código
             específico
             de P1
             Restaurar estado
             RETI

Rut_Espec_P2 Salvar estado
             Código
             específico
             de P2
             Restaurar estado
             RETI
             ...
```


Análisis

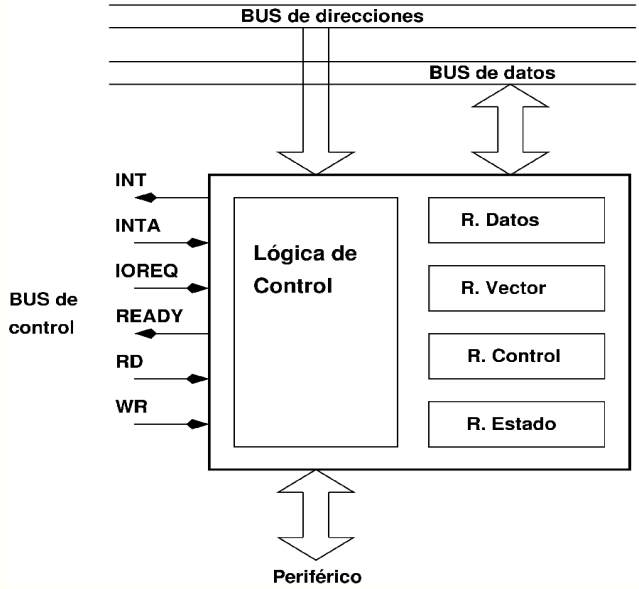
- En caso de peticiones simultáneas solo se ejecuta la más prioritaria que es la que se consulta primero
- No es posible el anidamiento entre las distintas rutinas de servicio porque la interrupción ha de ser previamente reconocida para averiguar su prioridad
- Se utilizan muchas instrucciones en la identificación de la interrupción cuando existen muchos periféricos

Vectorización

- Mediante el ciclo de bus de reconocimiento de interrupciones la CPU pide que el peticionario de la interrupción se identifique.
- El identificador se carga al iniciar la operación en el registro del vector de interrupción del módulo de entra/salida.



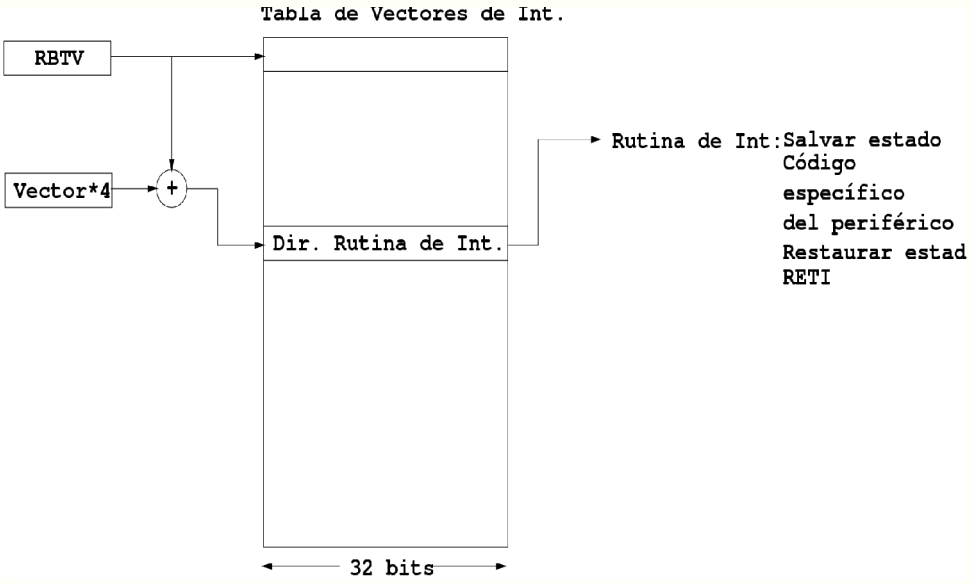
Módulo de entrada/salida con interrupciones vectorizadas



Secuencia de reconocimiento con vectorización

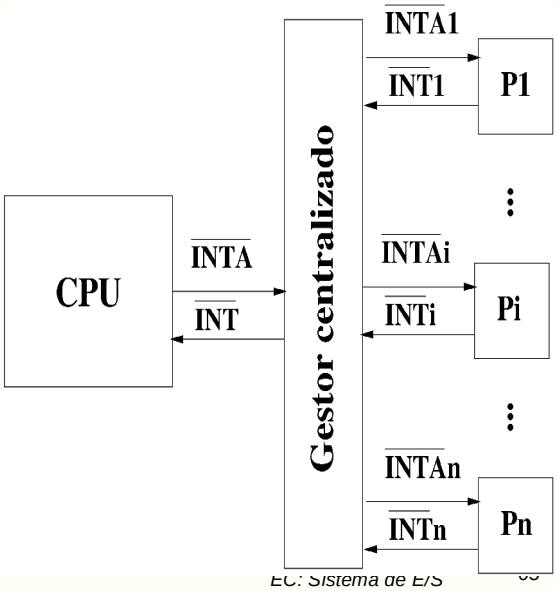
FETCH: Si $INT \wedge RE.\overline{BMI}$ entonces:
Salvar PC
Salvar RE
RE.BMI ← 1 (Inhibir interrupciones)
RE.S ← 1 (Cambiar a modo privilegiado)
Ciclo de bus de reconocimiento de interrupciones
(activación $\overline{INTA}$) DR ← Vector
PC ← f(Vector)
ir a *fetch*
Si no
ir a *fetch*

Tabla de vectores de interrupción

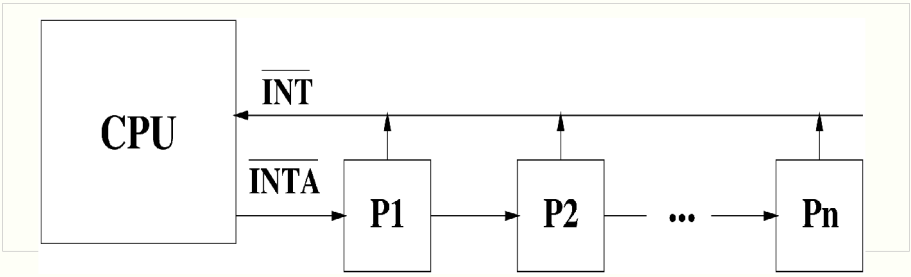


Prioridades

- La señal de reconocimiento le ha de llegar al más prioritario de los módulos solicitantes.
- Se necesita un esquema de prioridades hardware

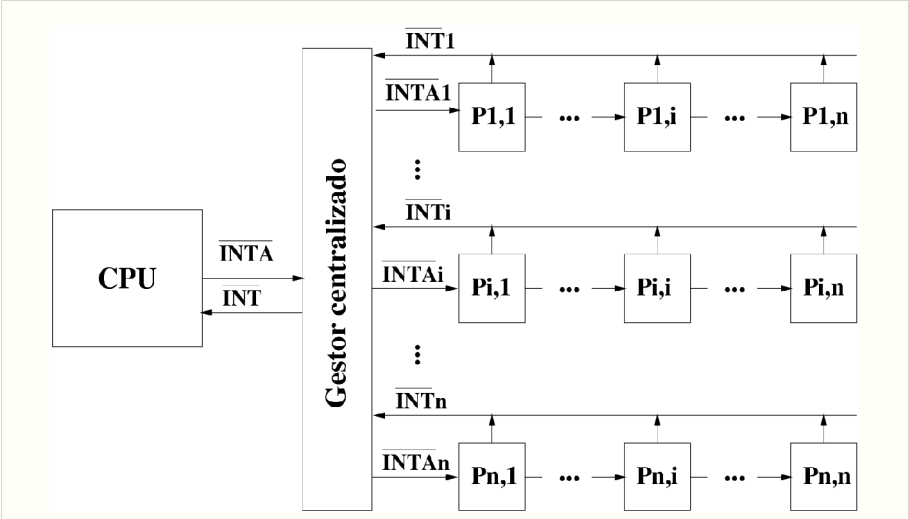


Gestor encadenado (*daisy chain*)



- El tiempo de ciclo de bus ha de ser suficiente
- Se pueden conectar tantos módulos como sea necesario

Híbrido



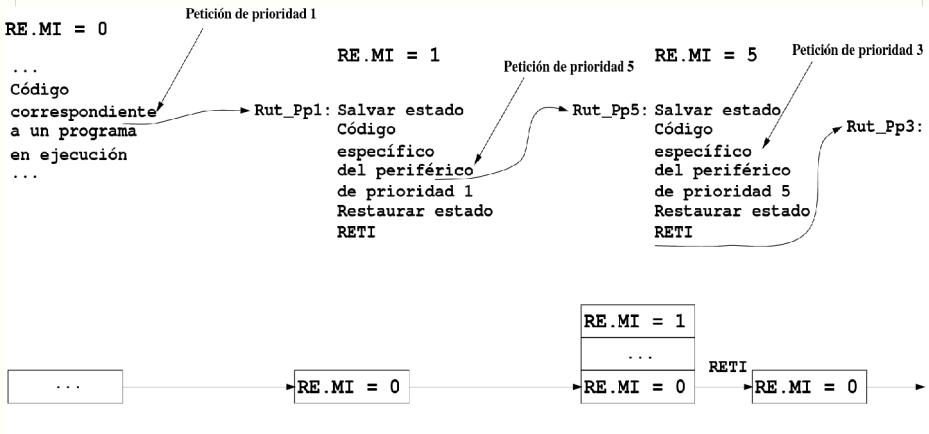
Análisis

- Rápida identificación del solicitante
- Conflictos de prioridades resueltos por hardware
- No es posible el anidamiento de rutinas de servicio:
  - Existe un solo bisestable de máscara con lo que o están todas permitidas o inhibidas
  - Con varios biestables de máscaras de interrupción se puede implementar una inhibición selectiva por niveles.

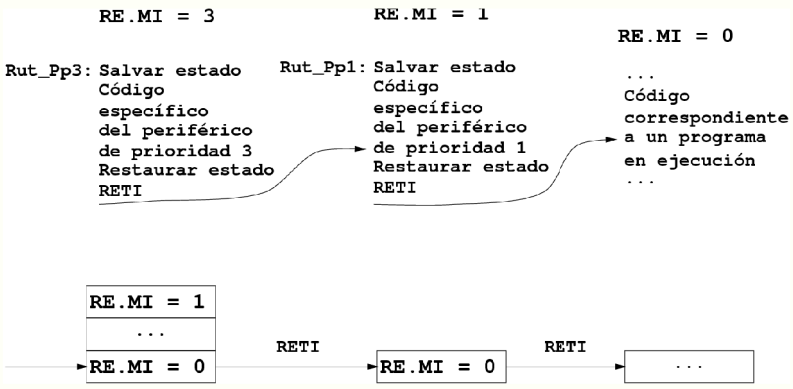
Secuencia de reconocimiento de interrupciones con inhibición selectiva

FETCH: Si $(i = max(INT_j), \forall j) \wedge (i > RE.MI)$ entonces:
Salvar PC
Salvar RE
RE.MI ← i (Establecer la nueva máscara de int.)
RE.S ← 1 (Cambiar a modo privilegiado)
Ciclo de bus de reconocimiento de interrupciones
(activación $\overline{INTA_i}$) DR ← Vector
PC ← f(Vector)
ir a *fetch*
Si no
ir a *fetch*

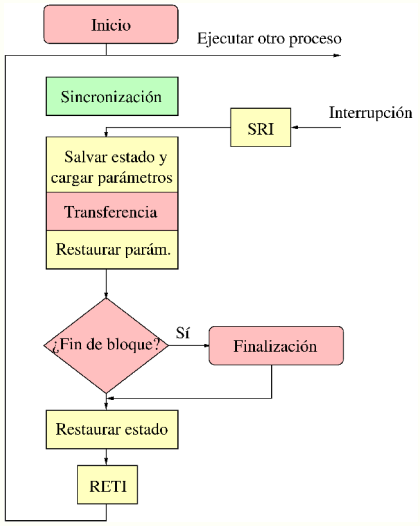
Ejemplo de anidamiento



Ejemplo de anidamiento (II)

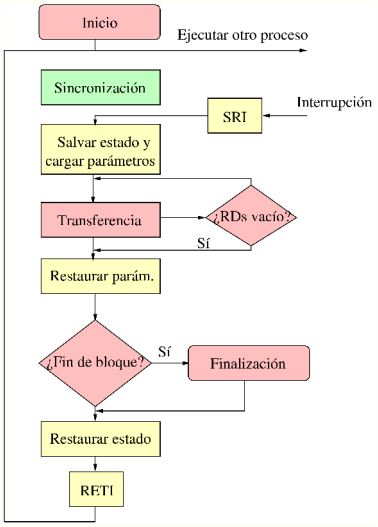


Análisis cuantitativo



- Se evita la sincronización pero se realizan otras operaciones para llevar a cabo la transferencia.
- El total supone mucho menos tiempo de CPU que por programa pero aún existe una sobrecarga inevitable.
- Para minimizar el impacto de esta sobrecarga se puede aumentar el tamaño del registro de datos.

Uso de *buffer*



- Se dota al módulo de entrada/ salida de un *buffer* de registros de datos
- Solicita la interrupción cuando el *buffer* está lleno (operación de entrada) o vacío (operación de salida)
- El número de interrupciones por operación se reduce y la sobrecarga por cada dato es mucho menor en función del tamaño del *buffer*

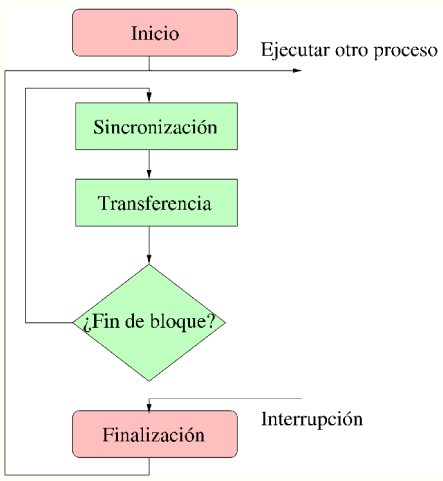
Asignación de prioridades

- El método óptimo consiste en asignar mayor prioridad al dispositivo que pide interrupciones con mayor frecuencia.
- La frecuencia depende de la velocidad de transferencia y del tamaño del registro de datos.
- Algunos dispositivos excepcionalmente no siguen esta regla:
  - Consola de operación
  - Temporizadores programables
  - DMA

Interrupciones no enmascarables

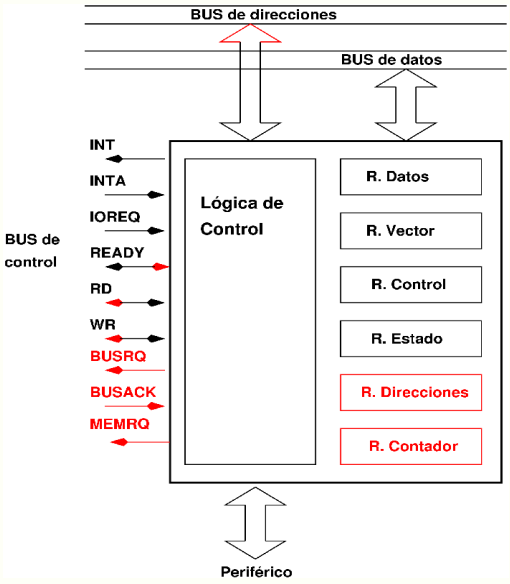
- Existen sucesos que no admiten demora en su tratamiento.
- Por ejemplo un fallo de energía.
- Para ellos existe una línea especial de petición de interrupción que no puede ser enmascarada:
  - NMI en la familia x86
  - INT_7 en la familia M68000

Entrada/salida mediante DMA



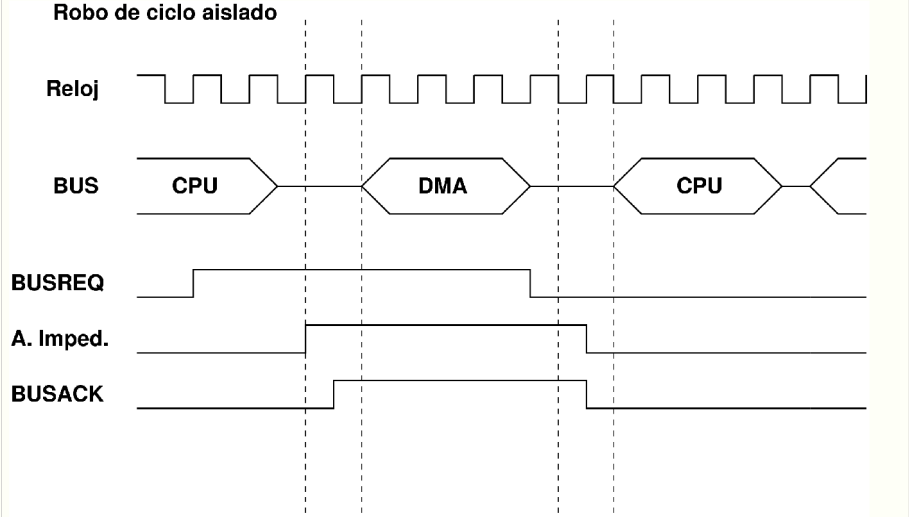
- La CPU se encarga de iniciar la operación.
- El módulo de entrada salida se encarga de la sincronización y transferencia y avisa cuando ha terminado.
- La CPU finaliza la operación.
- Hay una única interrupción por operación: se ahorra mucho tiempo de CPU con dispositivos de bloque.

DMA mediante robo de ciclo

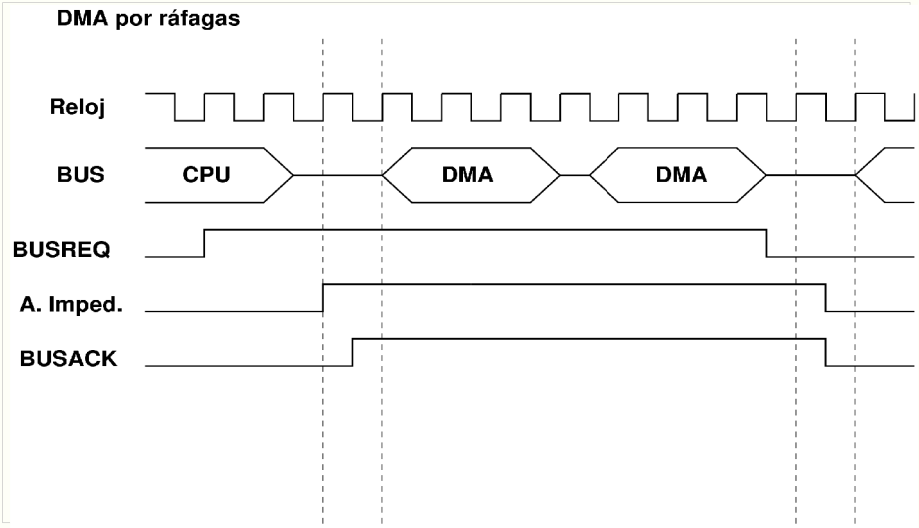


- El módulo solicita los buses con BUSRQ.
- La CPU los cede al final del ciclo de bus en curso indicándolo con BUSACK.
- La CPU se “desconecta” de los buses colocando sus salidas en alta impedancia.
- El módulo inicia el ciclo de bus para realizar la transferencia con memoria.
- Cuando acaba devuelve el bus desactivando BUSRQ.
- La CPU recupera los buses y desactiva BUSACK.

Robo de ciclo aislado (sin buffer)



Robo de ciclo en ráfagas (con buffer)



Operación por **DMA**

1.- **Programar** la operación de DMA en el periférico: instrucciones de salida (OUT)
1. Dir. Comienzo zona de Memoria: Dir. ⟶ DIR
2. Tamaño del bloque: Nº datos . ⟶ CONT
3. E ó S: E ó S ⟶ E/S

- A partir de ese momento la CPU se dedicará a "sus labores"

2.- Cuando el periférico está **listo** para la transferencia de un dato, **solicita los buses**:
Activa $\overline{\text{BUSREQ}}$

- La CPU "**cederá**" los **buses** cuando acabe la "fase" actual de la instrucción que está ejecutando:
  - "Congela" la ejecución de la instrucción (no ejecuta la siguiente Fase).
  - Pone en "alta impedancia" sus conexiones a:
    - Bus de Datos (**DB**)
    - Bus de Direcciones (**AB**)
    - Señales de control de Memoria ($\overline{\text{RD}}$, $\overline{\text{WR}}$, etc.)
  - Activa $\overline{\text{BUSACK}}$

3.- El periférico:
1. Transfiere el dato a/desde M activando las ss. de control y dirección pertinentes y usando el Bus de Datos.
2. CONT ⟵ CONT – 1
3. DIR ⟵ DIR + 1

4.- *Si* CONT ≠0 (* faltan más datos por transferir *)
entonces
Si perif. listo para transferir siguiente dato
entonces (*modo RÁFAGA*)
- Mantiene activa $\overline{\text{BUSREQ}}$
- GO TO **3**

si no (*robo de ciclo aislado*)
- Desactiva $\overline{\text{BUSREQ}}$
- la CPU:
  - desactiva $\overline{\text{BUSACK}}$
  - continúa ejecutando Fase siguiente
- GO TO **2**

Si no (* CONT = 0: bloque de datos transferido *)
- Desactiva $\overline{\text{BUSREQ}}$
- "avisa" a la CPU de que ha terminado la operación de DMA: solicita Interrupción:

Activa $\overline{\text{INT}}$

Diferencias y similitudes con las interrupciones

- Ambos son eventos asíncronos pero los robos de ciclo no alteran el estado de la CPU.
  - Por lo tanto se puede conceder en “cualquier” momento.
- Un ciclo de bus no es reanudable y muy breve.
  - Por lo tanto no se permite anidamiento.
- Para ambos mecanismos existe una línea para petición y otra para concesión:
  - Para peticiones de bus simultáneas se utilizan los esquemas de prioridades hardware: centralizado, daisy chain o híbrido.